Title	An $O(mn+n^2log n)$ Time Cactus Construction Algorithm (Mathematical Optimization Theory and its Algorithm)
Author(s)	Nagamochi, Hiroshi; Nakamura, Shuji; Ishii, Toshimasa
Citation	数理解析研究所講究録 (2001), 1241: 148-156
Issue Date	2001-12
URL	http://hdl.handle.net/2433/41638
Right	
Type	Departmental Bulletin Paper
Textversion	publisher

数理解析研究所講究録 1241 巻 2001 年 148-156

An $O(mn + n^2 \log n)$ Time Cactus Construction Algorithm

永持 仁　中村 秀司　石井 利昌
豊橋技術科学大学 大学院工学研究科 情報工学系

Hiroshi Nagamochi　Shuji Nakamura　Toshimasa Ishii
Department of Information and Computer Sciences,
Toyohashi University of Technology,
Toyohashi, Aichi, 441-8580, Japan.
{naga,shuji,ishii}@algo.ics.tut.ac.jp

Abstract

It is known that all minimum cuts in an edge-weighted, undirected graph can be represented by a *cactus*. In this paper, we show that such a cactus representation can be computed in $O(mn+n^2 \log n)$ time and $O(m)$ space. This improves the previous best time bound of deterministic cactus construction algorithms, and matches with the time bound of the fastest deterministic algorithm for computing a minimum cut.

1 Introduction

We consider an undirected graph G with n vertices and m edges. The minimum cut in a graph is one of the most fundamental notions in graph theory and is a rich source of interesting combinatorial problems. A connected graph is called a cactus if each edge is contained in exactly one cycle. Dinits, Karzanov and Lomonosov [3] showed that all minimum cuts in a given graph G can be represented as a cactus $\mathcal{R}$ of size $O(n)$ (see section 3 for the definition of the representation). Such a cactus $\mathcal{R}$ is called a cactus representation for all minimum cuts. This cactus representation plays an important role in solving many connectivity problems such as the edge-connectivity augmentation problem [4, 11] and the edge-splitting problem [9]. Several efficient algorithms for cactus representations have been developed so far. Currently an $O(nm \log(n^2/m))$ time algorithm due to Gabow [4] and an $O(mn + n^2 \log n + \gamma m \log n)$ time algorithm due to Nagamochi, Nakao and Ibaraki [10] are the fastest among these deterministic algorithms, where γ is the number of cycles in a cactus $\mathcal{R}$ and $\gamma = O(n)$ holds. An $\tilde{O}(n^2)$ time randomized algorithm of Monte Carlo type is proposed by Benczúr [2].

However, the fastest deterministic algorithm for computing the minimum cut size due to Nagamochi and Ibaraki [6] runs in $O(mn+n^2 \log n)$ time and $O(m)$ space. Thus, it was a little gap between the time complexities for computing a single minimum cut and for computing all minimum cuts. In this paper, we try to improve the complexity of Nagamochi, Nakao and Ibaraki's algorithm (NNI algorithm, for short), whose time and space complexities are $O(mn + n^2 \log n + \gamma m \log n) = O(mn \log n)$ and $O(mn)$, respectively. We show that the NNI algorithm can be implemented to run in $O(mn + n^2 \log n)$ time and $O(m)$ space. This closes the gap between the complexities of the NNI algorithm and of the Nagamochi and Ibaraki's minimum cut algorithm.

2 Preliminaries

Let G be a simple undirected graph with a set $V(G)$ of vertices and a set $E(G)$ of edges weighted by $c_G : E(G) \to R^+$, where R^+ is the set of non-negative real numbers. An edge in

$E(G)$ with end vertices u and v is denoted by (u, v) or (v, u). We denote $|V(G)|$ by n, $|E(G)|$ by m, and $V(G) - X$ by $\overline{X}$ for a subset $X \subseteq V(G)$. A graph G is said to be unweighted if $c_G(e) = 1$ for all edges $e \in E(G)$.

For two non-empty disjoint subsets $X, Y \in V$, we denote by $E(X, Y; G)$ the set of edges $e = (x, y)$ such that $x \in X$ and $y \in Y$, and also denote $c(X, Y; G) = \sum_{e \in E(X,Y;G)} c_G(e)$. We may write $E(X, Y; G)$ and $c(X, Y; G)$ simply as $E(X, Y)$ and $c(X, Y)$, respectively, if G is clear from the context.

A *partition* of $V(G)$ is a family $\{V_1, V_2, \ldots, V_r\}$ of non-empty subsets of $V(G)$ (possibly $r = 1$) such that any two subsets are pairwise disjoint and the union of all subsets is $V(G)$. An ordered set $(V_1, V_2, \ldots, V_r)$, $r \geqslant 2$, is called an *ordered partition* (or *o-partition*, for short) of $V(G)$ if $\{V_1, \ldots, V_r\}$ is a partition of $V(G)$. A partition $\{X, V(G) - X\}$ of $V(G)$ is called a *cut* of G, and its size is defined by $c(X, \overline{X})$. A cut $\{X, \overline{X}\}$ *crosses* another cut $\{Y, \overline{Y}\}$ in a graph G if

$$X \cap Y \neq \emptyset,\ X - Y \neq \emptyset,\ Y - X \neq \emptyset \text{ and } \overline{X \cup Y} \neq \emptyset.$$

A cut separates $x, y \in V(G)$ is called an (x, y)-*cut*. An (x, y)-cut with minimum size called a *minimum* (x, y)-*cut*, and its size is defined by $\lambda(x, y; G)$. A total ordering $v_1, v_2, \ldots, v_n$ of all vertices in $V(G)$ *maximum adjacency ordering* (*MAO*, for short) if it satisfies

$$c(\{v_1, v_2, \ldots, v_i\}, v_{i+1}) = \max_{u \in \{v_{i+1}, \ldots, v_n\}} c(\{v_1, v_2, \ldots, v_i\}, u) \quad (1 \leqslant i \leqslant n-1).$$

Lemma 2.1 [6, 7, 13] *For an MAO* $v_1, v_2, \ldots, v_n$, $\lambda(v_{n-1}, v_n; G) = c(\{v_n\}, V(G) - \{v_n\})$ *holds for the last two vertices* v_{n-1} *and* v_n. *An MAO in a graph* G *with* n *vertices and* m *edges can be found in* $O(m + n \log n)$ *time.* □

Lemma 2.2 [7] *For a given MAO in a graph* G *with* n *vertices and* m *edges, a maximum flow between the last two vertices* v_{n-1} *and* v_n *can be computed in* $O(m \log n)$ *time.* □

A cut with the minimum size is called a *minimum cut*, and its size is defined by $\lambda(G)$. We denote the set of all minimum cuts in G by $\mathcal{C}(G)$. For example, the graph G in Fig. 1 has the minimum cut size 4.

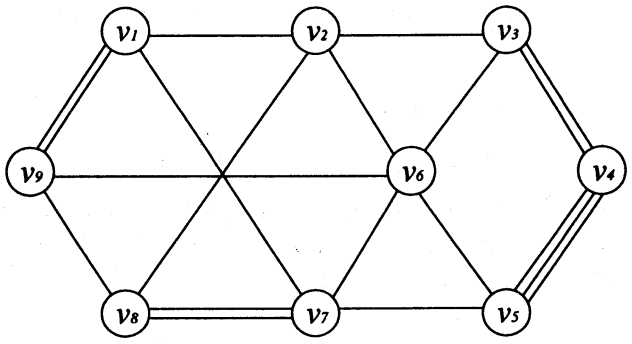

Figure 1: An edge-weighted undirected graph G with $\lambda(G) = 4$. (the number of lines between two vertices u and v represents weight of the edge (u, v)).

3 Cactus representation for minimum cuts

In this section, we review the definition of cactus representations for all minimum cuts and some basic operations for constructing cactus representations.

3.1 Definition of cactus representations

We call a graph consisting of a single vertex a *trivial* cactus. An unweighted graph with more than one vertex is called a *cactus* if each edge belongs to exactly one cycle, where every cycle is of length at least 2 (i.e., there is no self-loop). Thus, every pair of cycles, if any, in a cactus has at most one vertex in common. Any non-trivial unweighted cactus $\mathcal{R}$ satisfies $\lambda(\mathcal{R}) = 2$.

For a given graph G, we introduce an unweighted cactus $\mathcal{R}$ and a mapping $\varphi : V(G) \to V(\mathcal{R})$. Throughout this paper, we shall use the term "vertex" to denote an element in $V(G)$, and the term "node" to denote an element in $V(\mathcal{R})$. A set $V(\mathcal{R})$ may contain a node x such that $V(G)$ contains no vertex v with $\varphi(v) = x$. Such a node x is called an *empty node*. Let $\mathcal{C}(\mathcal{R})$ denote the set of all minimum cuts of $\mathcal{R}$. Thus, $\{S, V(\mathcal{R}) - S\} \in \mathcal{C}(\mathcal{R})$ holds if and only if $E(S, V(\mathcal{R}) - S; \mathcal{R})$ is a set of two edges belonging to the same cycle in $\mathcal{R}$.

Definition 3.1 *For a given subset $\mathcal{C}' \subseteq \mathcal{C}(G)$ of minimum cuts, a pair $(\mathcal{R}, \varphi)$ of a cactus $\mathcal{R}$ and a mapping φ is called a cactus representation for $\mathcal{C}'$ if it satisfies* (i) *and* (ii).

(i) *For an arbitrary minimum cut $\{S, V(\mathcal{R}) - S\} \in \mathcal{C}(\mathcal{R})$, the cut $\{X, \overline{X}\}$ defined by $X = \{u \in V(G) \mid \varphi(u) \in S\}$ and $\overline{X} = \{u \in V \mid \varphi(u) \in V(\mathcal{R}) - S\}$ belong to $\mathcal{C}'$*

(ii) *Conversely, for any minimum cut $\{X, \overline{X}\} \in \mathcal{C}'$, there exists a minimum cut $\{S, V(\mathcal{R}) - S\} \in \mathcal{C}(\mathcal{R})$ such that $X = \{u \in V \mid \varphi(u) \in S\}$ and $\overline{X} = \{u \in V \mid \varphi(u) \in V(\mathcal{R}) - S\}$.* □

It is shown in [3] that, for any graph G, there exists a cactus representation for $\mathcal{C}(G)$. For example, Fig. 2 shows a cactus representation for all minimum cuts in the graph in Fig. 1.

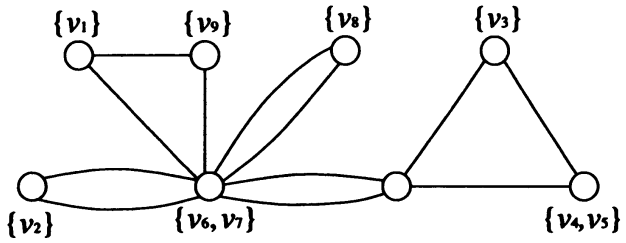

Figure 2: A cactus representation $\mathcal{R}$ for $\mathcal{C}(G)$ of the graph G in Fig. 1

3.2 Union of cactus representations

Suppose that we are given two cactus representations $(\mathcal{R}_1, \varphi_1)$ and $(\mathcal{R}_2, \varphi_2)$ for subset of minimum cuts $\mathcal{C}_1, \mathcal{C}_2 \in \mathcal{C}(G)$. We review an operation for combining $(\mathcal{R}_1, \varphi_1)$ and $(\mathcal{R}_2, \varphi_2)$ into a single cactus representation $(\mathcal{R}, \varphi)$ for $\mathcal{C}_1 \cup \mathcal{C}_2$. It is known [8] that such $(\mathcal{R}, \varphi)$ exists if there are nodes $z_1 \in V(\mathcal{R}_1)$ and $z_2 \in V(\mathcal{R}_2)$ such that

$$\varphi^{-1}(z_1) \cup \varphi^{-1}(z_2) = V(G).$$

The desired cactus $\mathcal{R}$ is produced by identifying the node z_1 with z_2, which is newly denoted as z. The node z is a cut point in $\mathcal{R}$. The mapping $\varphi : V(G) \to V(\mathcal{R}_1) \cup V(\mathcal{R}_2) \cup \{z\} - \{z_1, z_2\}$ is obtained by setting

$$\begin{aligned} \varphi^{-1}(z) &\equiv \varphi_1^{-1}(z_1) \cap \varphi_2^{-1}(z_2), \\ \varphi^{-1}(x_1) &\equiv \varphi_1^{-1}(x_1) \text{ for all nodes } x_1 \in V(\mathcal{R}) - z_1, \\ \varphi^{-1}(x_2) &\equiv \varphi_2^{-1}(x_2) \text{ for all nodes } x_2 \in V(\mathcal{R}) - z_2. \end{aligned}$$

The node z is called a *joint node* in the operation. The joint node z is an empty node in $(\mathcal{R}, \varphi)$ if and only if $\varphi_1^{-1}(z_1) \cap \varphi_2^{-1}(z_2) = \emptyset$ holds.

3.3 (s,t)-MC-partition

The following lemma is the basis for subsequent discussions.

Lemma 3.1 [3] *Let $\{X, \overline{X}\}$ and $\{Y, \overline{Y}\}$ be any two minimum cuts of G. If these cuts cross each other, then*

(i) $c(V_1, V_2) = c(V_2, V_3) = c(V_3, V_4) = c(V_4, V_1) = \lambda(G)/2$

(ii) $c(V_1, V_3) = c(V_2, V_4) = 0$

hold, where $V_1 = X \cap Y$, $V_2 = \overline{X} \cap Y$, $V_3 = \overline{X} \cap \overline{Y}$ and $V_4 = X \cap \overline{Y}$. □

The lemma says that for an edge $e = (s,t)$ with $c_G(e) > 0$ any two minimum cuts separating s and t do not cross each other (since otherwise $c_G(e) > 0$ would contradict $c(V_1, V_3) = 0$ or $c(V_2, V_4) = 0$).

Given an o-partition $(V_1, \ldots, V_r)$ and two indices h and k $(1 \leqslant h \leqslant k \leqslant r)$, we define

$$V_{(h,k)} \equiv V_h \cup V_{h+1} \cup \cdots \cup V_k.$$

For a subset $\mathcal{C}' \subseteq \mathcal{C}(G)$, an o-partition $(V_1, \ldots, V_r)$ of $V(G)$ is called a *minimum cut o-partition* (or *MC-partition*, for short) over $\mathcal{C}'$, if

$$\{\{V_{(1,k)}, \overline{V_{(1,k)}}\} \mid 1 \leqslant k \leqslant r-1\} \subseteq \mathcal{C}'.$$

We say that an edge $e = (s,t)$ in G is *critical* if $c_G(e) > 0$ and $\lambda(s,t;G) = \lambda(G)$. Let $\mathcal{C}_{(s,t)}(G)$ denote the set of all minimum cuts in $\mathcal{C}(G)$ that separate s and t. It is shown that set $\mathcal{C}_{(s,t)}(G)$ for a critical edge (s,t) has the following structure.

Lemma 3.2 [5, 12] *For a critical edge (s,t) in a graph G, all minimum cuts in $\mathcal{C}_{(s,t)}(G)$ are represented by an MC-partition over $\mathcal{C}(G)$.* □

An MC-partition as in the lemma is called an *(s,t)-MC-partition* over $\mathcal{C}(G)$, and is denoted by $\pi_{(s,t)}$. It is also known [5, 12] that a $\pi_{(s,t)}$ can be obtained in $O(m+n)$ time from an arbitrary maximum flow between s and t.

For example, Fig 3 shows the (v_2, v_3)-MC-partition $\pi_{(v_2,v_3)}$ of the graph G in Fig. 1.

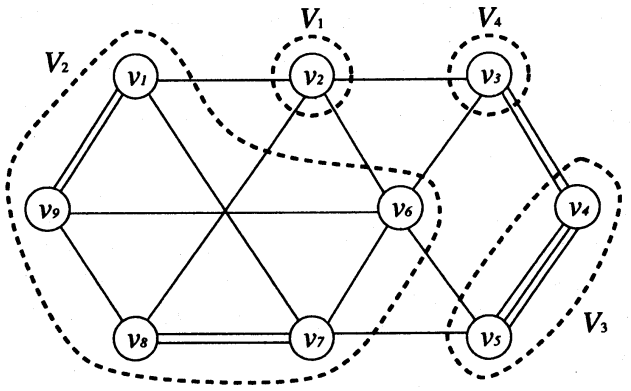

Figure 3: The (v_2, v_3)-MC-partition $\pi_{(v_2,v_3)}$ over $\mathcal{C}(G)$ for the graph G in Fig. 1

3.4 (s,t)-cactus representation

A cut $\{X, \overline{X}\}$ *crosses* an o-partition $(V_1, \ldots, V_r)$ of $V(G)$ if $\{X, \overline{X}\}$ crosses some cut of the form $\{V_i, \overline{V_i}\}$.

Lemma 3.3 [8] *Let (s,t) be a critical edge in a graph G. Then no minimum cut $\{X, \overline{X}\} \in \mathcal{C}(G)$ crosses the (s,t)-MC-partition $\pi_{(s,t)}$ over $\mathcal{C}(G)$.* □

We say that a cut $\{X, \overline{X}\}$ is *compatible* with an o-partition $\pi = (V_1, \ldots, V_r)$ of $V(G)$, if

$$V_i \subseteq X \text{ or } V_i \subseteq \overline{X} \text{ for all } i = 1, 2, \ldots, r.$$

We denote by $\mathcal{C}_{comp}(\pi)$ the set of all minimum cuts in $\mathcal{C}(G)$ that are compatible with π. For an (s,t)-MC-partition $\pi_{(s,t)} = (V_1, \ldots, V_r)$, Lemma 3.3 claims that any minimum cut $\{X, \overline{X}\} \in \mathcal{C}(G)$ satisfies either $\{X, \overline{X}\} \in \mathcal{C}_{comp}(\pi_{(s,t)})$ or $X \subset V_i$ for some i. Note that $\mathcal{C}_{comp}(\pi_{(s,t)})$ contains the set $\mathcal{C}_{(s,t)}(G)$ of all minimum cuts that separate s and t, and possibly some other minimum cuts. The next lemma shows how to compute all minimum cuts in $\mathcal{C}_{comp}(\pi_{(s,t)})$.

Lemma 3.4 [8] *For a critical edge (s,t) in a graph G, let $\pi_{(s,t)} = (V_1, \ldots, V_r)$ be the (s,t)-MC-partition over $\mathcal{C}(G)$. Then, any minimum cut $\{X, \overline{X}\} \in \mathcal{C}_{comp}(\pi_{(s,t)})$ satisfies the following:*

(i) *If $\{X, \overline{X}\}$ separates s and t, then either $X = V_{(i,r)}$ or $\overline{X} = V_{(i,r)}$ for some i $(1 < i \leqslant r)$.*

(ii) *If $\{X, \overline{X}\}$ does not separate s and t, then $X = V_{(i,j)}$ or $\overline{X} = V_{(i,j)}$ for some i, j $(1 < i \leqslant j < r)$ such that $\{V_k, \overline{V_k}\} \in \mathcal{C}(G)$ for all k $(i \leqslant k \leqslant j)$.* □

It is known [8] that any (s,t)-MC-partition $\pi_{(s,t)}$ for a critical edge (s,t) in G admits a cactus representation for $\mathcal{C}_{comp}(\pi_{(s,t)}) \in \mathcal{C}(G)$, which is called an *$(s,t)$-cactus representation* and denoted by $(\mathcal{R}_{(s,t)}, \varphi_{(s,t)})$. A linear time algorithm for constructing such a representation is also known.

Theorem 3.1 [8] *Given an (s,t)-MC-partition $\pi_{(s,t)}$ for a critical edge (s,t) in a graph G, a (s,t)-cactus representation $(\mathcal{R}_{(s,t)}, \varphi_{(s,t)})$ can be constructed in $O(m+n)$ time.* □

4 Algorithm for Constructing Cactus Representations

In this section, we review an outline of NNI algorithm for constructing a cactus representation for $\mathcal{C}(G)$, and then improve its time and space complexities.

4.1 Decomposition of a graph

For an (s,t)-MC-partition $\pi_{(s,t)} = (V_1, \ldots, V_r)$ and (s,t)-cactus representation $(\mathcal{R}_{(s,t)}, \varphi_{(s,t)})$ in G, let G_i be the graph constructed from G by contracting $V(G) - V_i$ $(i = 1, 2, \ldots, r)$ into a single vertex $\hat{v}_i$. Obviously, $\lambda(G_i) \geqslant \lambda(G)$. The graph G_i is called *critical* if $\lambda(G_i) = \lambda(G)$. Assume that G_i is critical, and consider a minimum cut $\{Y, V(G_i) - Y\} \in \mathcal{C}(G_i)$. By Lemma 3.4, either

(i) $\{Y, V(G_i) - Y\}$ separates some two vertices x and y in V_i and satisfies $Y \subset V_i$ (or $V(G_i) - Y \subset V_i$), or

(ii) $\{Y, V(G_i) - Y\} = \{V_i, \hat{v}_i\}$.

Note that the cut $\{V_i, \hat{v}_i\}$ in (ii) is already represented in $(\mathcal{R}_{(s,t)}, \varphi_{(s,t)})$ as $\{V_i, V(G) - V_i\}$. The ordered collection $(G_1, G_2, \ldots, G_r)$ of graph is called an (s,t)-*decomposition* of G. For every critical graph G_i, we try to compute a cactus representation $(\mathcal{R}_{G_i}, \varphi_{G_i})$ for $\mathcal{C}(G_i)$ or for $\mathcal{C}(G_i) - \{V_i, \hat{v}_i\}$. Thus every minimum cut in G is represented in $(\mathcal{R}_{(s,t)}, \varphi_{(s,t)})$ or $(\mathcal{R}_{G_i}, \varphi_{G_i})$ for some i.

4.2 NNI algorithm

From the argument in the previous section, the outline of NNI algorithm is described recursively as follows.

Procedure CACTUS(G)
Input: a graph G.
Output: a cactus representation $(\mathcal{R}, \varphi)$ for all minimum cuts $\mathcal{C}(G)$.

1. Choose an edge $e = (s,t)$ with $c_G(e) > 0$ in G.
2. If $\lambda(s,t;G) > \lambda(G)$, then contract s and t into a single vertex, and apply Procedure CACTUS(G') to the resulting network G'.
3. Otherwise (if $\lambda(s,t;G) = \lambda(G)$), compute a maximum flow between s and t.
4. Compute the (s,t)-MC-partition $\pi_{(s,t)} = (V_1, \ldots, V_r)$ and the corresponding (s,t)-cactus representation $(\mathcal{R}_{(s,t)}, \varphi_{(s,t)})$.
5. If all minimum cuts in the $\pi_{(s,t)} = (V_1, \ldots, V_r)$ have been already detected then contract s and t and return a trivial cactus.
6. Otherwise, find the (s,t)-decomposition $(G_1, G_2, \ldots, G_r)$ of G. Go to 7.
7. Apply Procedure CACTUS(G_i) to each G_i to obtain the cactus representation $(\mathcal{R}_{G_i}, \varphi_{G_i})$ for $\mathcal{C}(G_i)$, where Procedure CACTUS(G_i) returns a trivial cactus if G_i is not critical. Go to 8.
8. Combine all $(\mathcal{R}_{G_i}, \varphi_{G_i})$, $i = 1, \ldots, r$ and $(\mathcal{R}_{(s,t)}, \varphi_{(s,t)})$ into a cactus representation $(\mathcal{R}, \varphi)$ for $\mathcal{C}(G)$ by using union operations. Return $(\mathcal{R}, \varphi)$.

From the argument so far, we see that the above algorithm computes a cactus representation for $\mathcal{C}(G)$. We use an MAO to choose an edge $e = (s,t)$ and to compute a maximum flow between s and t. From an MAO $v_1, v_2, \ldots, v_n$, we choose the vertex v_p with the largest index p such that v_p and v_n are joined by an edge with positive weight. Then let $s = v_n$ and $t = v_p$. Note that $v_1, v_2, \ldots, v_p, v_n$ is an MAO in the graph G' obtained from G by deleting vertices $v_{p+1}, v_{p+2}, \ldots, v_{n-1}$. Hence, by Lemma 2.1, $\lambda(s,t;G) = \lambda(s,t;G') = c(\{s\}, V(G') - \{s\})$ holds, and by Lemma 2.2 a maximum flow between s and t can be found in $O(m \log n)$ time. After computing a maximum flow between s and t in Step 3, the (s,t)-MC-partition $\pi_{(s,t)}$ and the (s,t)-cactus representation $(\mathcal{R}_{(s,t)}, \varphi_{(s,t)})$ can be computed in linear time.

The entire time complexity of CACTUS is dominated by the time for finding MAOs and for computing maximum flows. As to these time complexities, the following result is shown.

Lemma 4.1 [10] *The total time to compute all MAO during the execution of CACTUS is* $O(mn + n^2 \log n)$. *Also, the total time to compute all maximum flows during the execution of CACTUS is* $O(n \cdot M(m,n))$, *where* $M(n,m)$ *denotes the time to compute a maximum flow between two adjacent vertices* s *and* t, *which are allowed to be chosen arbitrarily in the given graph.* □

Since we have seen $M(n,m) = O(m \log n)$, the time complexity of CACTUS is $O(nm \log n)$ (note that $M(n,m)$ does not include the time for choosing adequate s and t). Nagamochi, Nakao and Ibaraki reduced the number of iterations of maximum flow computations to improve

the time complexity to $O(mn + n^2 \log n + \gamma m \log n)$, where γ is the number of cycles in the resulting cactus representation. To improve the time complexity, we here use the following recent result by Arikati and Mehlhorn's maximum flow algorithm [1].

Lemma 4.2 [1] *A maximum flow between the last two vertices v_{n-1} and v_n of MAO can be computed in $O(m)$ time.* □

Hence, for $s = v_n$ and $t = v_p$ in an MAO, we can find a maximum flow in $O(m)$ time. Thus, $M(n, m) = O(m)$. Therefore, by Lemma 4.1, CACTUS can be implemented to run in $O(mn + n^2 \log n)$ time.

We next consider the space complexity of CACTUS. We use an adjacency list for a data structure to store a graph, which takes $O(m + n)$ space to store a graph with n vertices and m edges. We see that a naive implementation of CACTUS takes $O(mn)$ space. Indeed, in Step 7, procedure CACTUS(G_i) needs information of G_i and one may make a copy of G_i as an input of CACTUS(G_i), which will be maintained until the recursive process of CACTUS(G_i) is done. Notice that at the same depth of recursive calls, the total size of G_i is $O(n+m)$ because each G_i is obtained from G by contracting $\overline{V_i}$ to single vertex $\hat{v}_i$. However, since the depth of recursive call of Procedure CACTUS is $O(n)$, the space complexity of this implementation is $O(mn)$.

We improve this to $O(m)$ by introducing a data structure for handling contraction of vertices. Let us consider a data structure by which we can easily restore G from each G_i after procedure CACTUS(G_i) is finished. A graph G with n vertices is stored by n adjacency lists $L(u)$, $u \in V$, such that each element of the list $L(u)$ corresponds to an edge (u, v). Note that the same edge (v, u) is also stored in the adjacency list $L(v)$. Then an edge (u, v) can be viewed as a pair of directed edges (u, v) and (u, v) oppositely oriented. This is because if we remove the element (u, v) from $L(u)$ then we can traverse e from v to u but not from u to v. We use this property to restore G from G_i.

In order to restore G from G_i, we need to know which vertices are contracted and which edges are deleted in constructing G_i from G. Assume that we have just constructed a G_i from G by contracting $\overline{V_i}$ into a single vertex $\hat{v}_i$ in Step 6. For a newly created vertex $\hat{v}_i$, we construct an adjacency list $L(\hat{v}_i)$ which contains edges between $\hat{v}_i$ and $V(G) - \overline{V_i}$. As to edges $e = (u, v)$ between a vertex $v \in \overline{V_i}$ and a vertex $u \in V_i$, we remove e from $L(u)$ (but not from $L(v)$) so that we cannot traverse e from u to v (but we can still traverse e from v to u, although this property is not necessarily during execution of CACTUS(G_i)). Note that in the G_i we cannot reach any of vertices in $\overline{V_i}$, and the existence of such edges in the list $L(v)$, $v \in \overline{V_i}$ does not affect the subsequent computation of the algorithm. Notice that we only remove edges $e = (u, v)$ with $u \in V_i$ and $v \in \overline{V_i}$ from $L(v)$ (hence edges whose end vertices are in $\overline{V_i}$ are not deleted). Let $E_i(G)$ denote the set of these edges. To store the information of $\overline{V_i}$, we construct a linear list $P(\hat{v}_i)$ consisting of $\hat{v}_i$ and $\overline{V_i}$. With $P(\hat{v}_i)$ and edges in the lists $L(v)$ for $v \in \overline{V_i}$, we can easily reconstruct G from G_i. We first traverse $P(\hat{v}_i)$ to identify all vertices v contracted into $\hat{v}_i$ when G_i is constructed, and for each $v \in \overline{V_i}$, we traverse $L(v)$ to identify all edges e incident to v. If an edge $e = (v, u)$ incident to $v \in \overline{V_i}$ joins v and a vertex $u \in V_i$, then we insert edge (u, v) in the list $L(u)$ (where such edge (u, v) must have been removed when G_i constructed). Therefore, to perform the recursive calls in CACTUS, we only need to maintain the linear lists $P(\hat{v}_i)$, whose size in total is $O(n)$ since each vertex in the input graph G appears in at most one linear list. Therefore, the space complexity of this implementation is $O(m + n)$.

From above discussion, the next theorem is established.

Theorem 4.1 *A cactus representation for all minimum cuts in a graph G can be constructed in $O(mn + n^2 \log n)$ time and $O(m + n)$ space.* □

5 Experiment

To evaluate the practical improvement of space complexity of algorithm CACTUS, we implemented the existing method and our new method and measured the maximum memory size used during execution of CACTUS. In our experiment, we used the following three different problems.

(a) Complete binary trees, where weight of each edge is 1.

(b) Union of k Hamilton cycles on n vertices (where each edge in a Hamilton cycle has weight 1, and the resulting graph satisfies $c(v, V - v) = k$ for all $v \in V$)

(c) Spherical surface graphs, which are graphs obtained as follows. Put n vertices randomly on a spherical surface, and create an edge between two vertices u and v if the central angle $\angle uOv$ is less than a threshold d (where O denotes the center of the sphere) Also, all edges have weight 1.

Tables 1,2 and 3 show results of the experiments for graphs (a),(b) and (c), where each data is the average over 30 instances:

Table 1: Complete binary tree

number of vertices	200	400	600	800	1000	1200
existing method (KByte)	3111	12165	27429	48316	75361	108577
proposal method (KByte)	160	319	484	642	804	964

Table 2: Union of Hamilton cycle ($n = 1000$)

k	2	3	4	5	6	7
existing method (KByte)	10954	2710	4976	5647	7558	3530
proposal method (KByte)	1007	1189	1396	1602	1810	2006

Table 3: Spherical surface graph ($n = 100$)

d	30	35	40	45	50	55
average of number of edges	313	461	568	717	920	1062
existing method (KByte)	142	160	190	207	251	291
proposal method (KByte)	124	148	175	202	236	270

In case of (a), the memory size is reduced drastically by our method. This is because the depth of recursive calls is large and hence the previous method requires large memory size. In case of (b) and (c), there are no big difference between the previous method and our method, since minimum cuts always appear around a single vertex and hence the depth for recursive calls is very small.

6 Conclusion

In this paper, we show that the complexity of NNI algorithm can be improved to $O(mn + n^2 \log n)$ time and $O(m + n)$ space by using Arikati and Mehlhorn's maximum flow algorithm and devising a data structure for graph contraction.

There are several interesting open problems. For example: (1) Can we construct a cactus representation for nearly minimum cuts of undirected graph? (2) Is there a simple representation for all minimum cuts of an other type of graph (directed graph, hypergraph, etc.)?

References

[1] S. R. Arikati and K. Mehlhorn, *A correctness certificate for the Stoer-Wagner min-cut algorithm*, Information Processing Letters, vol. 70, 1999, pp. 251-254.

[2] A. A. Benczúr, *Augmenting undirected connectivity in RNC and in randomized $\tilde{O}(n^3)$ time*, Proc. 26th ACM Symposium on theory of Computing, 1994, pp.658-667.

[3] E. A. Dinits, A. V. Karzanov and M. V. Lomonosov, *On the structure of a family of minimal weighted cuts in a graph*, Studies in Discrete Optimization (in Russian) A. A. Fridman (Ed.) Nauka, Moscow, 1976, pp. 290-306.

[4] H. N. Gabow, *A representation for crossing set families with applications to submodular flow problems*, Proc. 4th ACM Symposium on Discrete Algorithms, 1993, pp. 202-211.

[5] A. V. Karzanov and E. A. Timofeev, *Efficient algorithm for finding all minimal edge cuts of a nonoriented graph*, Kibernetika, vol. 2, 1984, pp. 8-12; translated in Cybernetics, 1986, pp. 156-162.

[6] H. Nagamochi and T. Ibaraki, *Computing edge-connectivity of multigraphs and capacitated graphs*, SIAM J. Disc. Math., vol. 5, 1992, pp. 54-66.

[7] H. Nagamochi, T. Ishii and T. Ibaraki, *A simple proof of a minimum cut algorithm and its applications*, Institute of Electronics, Information and Communication Engineers trans. fundamentals, vol E82-A, no. 10, 1999, pp. 2231-2236.

[8] H. Nagamochi and T. Kameda, *Constructing cactus representation for all minimum cuts in an undirected network*, Operations Research Society of Japan, vol. 39, 1996, pp. 135-158.

[9] H. Nagamochi, S. Nakamura and T. Ibaraki, *A simplified $\tilde{O}(nm)$ time edge-splitting algorithm in undirected graphs*, Algorithmica, vol.26, 2000, pp.56–67.

[10] H. Nagamochi, Y. Nakao and T. Ibaraki, *A fast algorithm for cactus representations of minimum cuts*, Journal of Japan Society for Industrial and Applied Mathematics, vol. 17, 2000, pp. 245-264.

[11] D. Naor, D. Gusfield and C. Martel, *A fast algorithm for optimally increasing the edge connectivity*, SIAM J. Computing, 26, 1997, pp.1139–1165.

[12] D. Naor and V. V. Vazirani, *Representing and enumerating edge connectivity cuts in RNC*, Proc. 2nd Workshop on Algorithms and Data Structures (F. Dehne, J. R. Sack and N. Santoro, eds.), Lecture Notes in Computer Science, 519, Springer Verlag, 1991, pp. 273-285.

[13] M. Stoer and F. Wagner: *A simple min cut algorithm*, JACM, 44, 1997, pp.585–591.